Claims Listing:

1. (original) A method for transforming test cases, comprising:

importing test cases written in one or more scripting languages;

converting test cases to an abstract representation that includes application state, external interaction sequences and input data; and

storing abstract representation of test cases into a database system.

2. (original) The method of claim 2, wherein an application state represents a runtime snapshot of application under test which defines the context of external interaction.

3. (original) The method of claim 2, wherein the application state includes a set of application objects, its attributes and attribute values.

4. (original) The method of claim 2, wherein the applications states corresponding to a test case are arranged in a hierarchical manner.

5. (original) The method of claim 2, wherein the database system is a relational database management system.

6. (original) The method of claim 2, wherein the database system is an XML database management system.

7. (original) The method of claim 2, wherein the scripting languages can be typed or untyped programming languages used for recording or authoring test cases.

8. (original) The method of claim 2, wherein the external interaction sequences represent events invoked by external agents on the application objects.

9. (original) The method of claim 8, wherein the external agents can be either human agents or other software agents.

10. (original) The method of claim 8, wherein the interaction sequencing includes flow control structures for capturing sequential, concurrent, looping and conditional interactions.

11. (original) The method of claim 2, further comprising:

implementing a syntax analyzer for incoming scripts.

12. (original) The method of claim 11, wherein the syntax analyzer is implemented one for each scripting language.

13. (original) The method of claim 12, wherein the syntax analyzer utilizes rules of syntax analysis that are specified in Extended Backus-Naur Form (EBNF).

14. (original) The method of claim 12, wherein the syntax analysis generates a parse tree in the form of an Abstract Syntax Tree (AST).

15. (original) The method of claim 2, further comprising:

implementing a semantic analysis that converts the abstract syntax tree to an abstract test case representation based on an Application Object Model (AOM).

16. (original) The method of claim 15, wherein the semantic analysis decomposes the test cases represented as an Abstract Syntax Tree into application state, external interaction sequences and input data.

17. (original) The method of claim 15, wherein an application object model is a metadata representation for modeling application under test.

18. (original) The method of claim 17, wherein the metadata representation includes object type definitions for application objects.

19. (original) The method of claim 17, wherein the metadata representation includes attribute definitions for each application object type.

20. (original) The method of claim 17, wherein the metadata representation includes definition of methods and events that are supported by each application object type.

21. (original) The method of claim 17, wherein the metadata representation includes definition of effects of events on an application state.

22. (original) The method of claim 18, wherein application object type definitions include additional categorization of each application object types into hierarchical, container and simple types.

23. (original) The method of claim 22, wherein the hierarchical object types are associated with an application state of its own; wherein application object types that can contain instances of other objects are termed as container types.

24. (original) The method of claim 23, wherein the state associated with a hierarchical application object type is a modal application state or a nonmodal application state.

25. (original) The method of claim 24, wherein a modal application state restricts possible interactions to application object instances available within the current application state.

26. (original) The method of claim 22, wherein the effects of events on an application state capture one or more consequences of the event to the application state.

27. (original) The method of claim 26, wherein a consequence of an event is selected from, creation of a new object instance of a given type, deletion of an object instance of a given type, modification of attributes of an existing object instance and selection of an instance of an object type.

28. (original) The method of claim 27, wherein creation of a new instance of an object of type that is hierarchical results in creation of a new application state.

29. (original) The method of claim 27, wherein selection of an object instance of type that is hierarchical results in selection of the application state associated with that object instance.

30. (original) The method of claim 2, further comprising:
enriching the abstract representation of test cases with information from an application metadata repository.

31. (original) The method of claim 30, wherein the enrichment of abstraction representation of test cases involves extracting values for those attributes of application objects associated with the test cases that are missing in the incoming test scripts.

32. (original) The method of claim 30, wherein enriching the abstraction representation of test cases includes decoupling of test cases from their recording or authoring environments.

33. (original) The method of claim 30, wherein enriching the abstraction representation of test cases allows usage of attributes that are stable within an application metadata representation.

34. (original) The method of claim 33, wherein using an identification field for a given object within the application metadata repository improves the reusability of a test case instead of a label used to represent the same object within a user interface which can change based on the locale of the application.

35. (original) The method of claim 33 wherein using an identification field allows to overcome the problem of different test execution environments using different attributes to identify the same application object.

36. (original) The method of claim 35, wherein enriching the abstraction representation of test cases enables representation of test cases that are test execution environment independent.

37. (original) The method of claim 2, further comprising:
separating application object attributes and input data from external interaction sequencing provides automatic parameterization.

38. (original) A system for transforming test cases, comprising:
a processor for importing test cases written in one or more scripting languages;
logic for converting test cases to an abstract representation that includes application state, external interaction sequences and input data; and a database that stores abstract representation of test cases.

39. (original) The system of claim 38, further comprising:
a syntax analyzer for incoming scripts.

40. (original) The system of claim 38, further comprising:
logic for implementing a semantic analysis that converts the abstract syntax tree to an abstract test case representation based on an Application Object Model (AOM).

41. (original) The system of claim 38, further comprising:

logic for selecting an object instance of type that is hierarchical results in selection of the application state associated with that object instance.

42. (original) The system of claim 38, further comprising:

logic for enriching the abstraction representation of test cases to enable representation of test cases that are test execution environment independent.

43. (original) A computer system, comprising:

a processor;

a memory coupled to the processor, the memory storing program instructions executable by the processor for converting test cases to an abstract representation that includes application state, external interaction sequences and input data; and

a database that stores abstract representation of test cases.

44. (original) The system of claim 43, further comprising:

a syntax analyzer for incoming scripts.

45. (original) The system of claim 44, wherein the syntax analyzer generates a parse tree in the form of an Abstract Syntax Tree (AST).

46. (original) The system of claim 45, further comprising:

logic to implement semantic analysis and convert the AST to an abstract test case representation based on an Application Object Model (AOM).

47. (original) The system of claim 43, further comprising:

logic for enriching the abstract test case representation with information from an application metadata repository.

48. (original) The system of claim 43, further comprising:

logic for separating application object attributes and input data from external interaction sequencing to provide automatic parameterization.

49. (original) A carrier medium, comprising:

program instructions for converting test cases to an abstract representation that includes application state, external interaction sequences and input data, and

a database for storing program instructions of abstract representations of test cases